Lab 17

Newton-Cotes Integration

Lab Objective: *Explain Newton-Cotes Integration*

One of the fundamental problems in calculus is integration. However, in the vast majority of cases it is impossible to integrate functions analytically. In these cases our only option is to approximate the integral numerically.

One of the most natural approaches is to use the trapezoidal rule. This approach approximates the integral of a function by using the average of its values at the left and right endpoints of the interval of integration:

$$\int_a^b f(x)dx \approx (b-a)\frac{f(a)+f(b)}{2}$$

This approach is equivalent to approximating the integral by integrating the linear interpolant of the function. For an illustration see Figure 17.1

A few questions rise naturally about the trapezoidal rule. The first is, how good is our approximation? It turns out, that for a twice differentiable function the error is equal to:

$$\text{error} = \frac{(b-a)^3}{12}f^{(2)}(\xi) \text{ where } \xi \in [a,b]$$

Thus, the error is proportional to both $(b-a)^3$ and the second derivative. The fact that the error is proportional to the value of the second derivative makes sense: the trapezoidal rule cannot compensate for a function having non-zero second derivative.

How can we more accurately approximate the integral of a function? One technique is to break our interval into many smaller sub-intervals. This is known as a *composite rule*. We will delay discussion of composite rules momentarily and discuss a different approach first.

Note that the trapezoidal rule is really just approximating the integral by integrating the linear interpolant of the function. Suppose instead that we approximate our function with a higher-order interpolant. We can write this method mathematically as follows:

$$\int_a^b f(x)dx \approx \int_a^b \sum_{j=1}^{n} L_j(x)f(x_j)dx = \sum_{j=1}^{n} f(x_j)\int_a^b L_j(x)dx$$

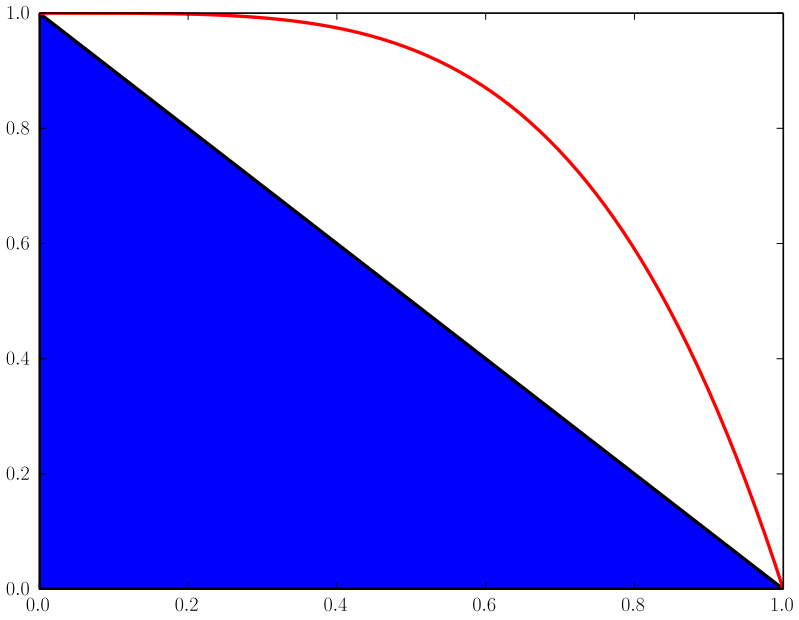

Figure 17.1: Demonstration of approximating the integral $\int_0^1 (1 - x^4)dx$ using the trapezoidal rule. The shaded area is the area approximated by the trapezoidal rule, and the actual function is given in red.

We can analytically pre-compute the value of $w_j = \int L_j(x)$ for a given set of interpolation points $\{x_j\}$ and thus write our approximation as

$$\int_a^b f(x)dx \approx \sum_j w_j f(x_j)$$

A set of points x_j and corresponding weights w_j are known as a *quadrature rule*.

Note that this method will exactly integrate polynomials of order $n - 1$ (where n is the number of points). This is because the polynomial interpolant ($\sum L_j f(x_j)$) will exactly match the function f.

Using three evenly-spaced points we can (using a little algebra) derive the following integration rule:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{6}\left(f(x_1) + 4f(x_2) + f(x_3)\right)$$

This is known as *Simpson's Rule*. We can implement Simpson's rule in Python as follows:

```
>>> def SimpsonsRule(func, a, b):
>>>     return ((b-a)/6.0)*sum(sp.array([1,4,1])*func(sp.array([a,(a+b)/2.0,b])))
```

We can test this function on $\sin(x)$ as follows:

```
>>> import scipy as sp
>>> SimpsonsRule(sp.sin, 0, sp.pi)
    2.0943951023931953
```

The answer, which we can find analytically, is 2. For only evaluating the function at three points this is fairly accurate.

Problem 1. Two higher-order quadrature rules are the Simpson 3/8 rule and Boole's rule. They can be written in the following formulae:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{8}\left(f(x_1) + 3f(x_2) + 3f(x_3) + f(x_4)\right)$$
$$\int_a^b f(x)dx \approx \frac{(b-a)}{90}\left(7f(x_1) + 32f(x_2) + 12f(x_3) + 32f(x_4) + 7f(x_5)\right)$$

Write functions that implements these quadrature rules.

Another tool of great importance in numerical integration is composite quadrature rules. These rules break up the interval into many smaller sub-intervals. A desired quadrature rule is then applied to each sub-interval, and the results are summed together. Mathematically we can write such an approach as follows:

$$\int_a^b f(x)dx = \sum_{i=0}^{n} \int_{x_i}^{x_{i+1}} f(x)dx$$

where $a = x_0 < x_1 < \ldots < x_n = b$. We then apply some rule (such as Simpson's rule, or the trapezoid rule) to each smaller integration problem. A graphical illustration of such an approach is shown in Figure 17.2

Let's look at how the composite Simpson's rule would work. We divide our interval of integration into n sub-intervals (and thus we have $n+1$ interval endpoints). To use Simpson's rule on each sub-interval we will need the n midpoints of each subinterval. Thus we have points $x_1 \ldots x_{2n+1}$. Of these points, the midpoints are precisely the even indices. Now recall that Simpson's Rule is:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{6}\left(f(x_1) + 4f(x_2) + f(x_3)\right)$$

Applying this to each sub-interval we get the following:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{6n}\left(f(x_1) + \sum_{i=1}^{n} 4 * f(x_{2i}) + \sum_{i=0}^{n-2} 2 * f(x_{1+2i}) + f(x_{2n+1})\right)$$

The initial constant is because each sub-interval has width $(b-a)/6n$. The different weights on each function evaluation $f(x_i)$ are justified as follows: each sub-interval midpoint gets weight of four by Simpson's rule, and sub-interval endpoints

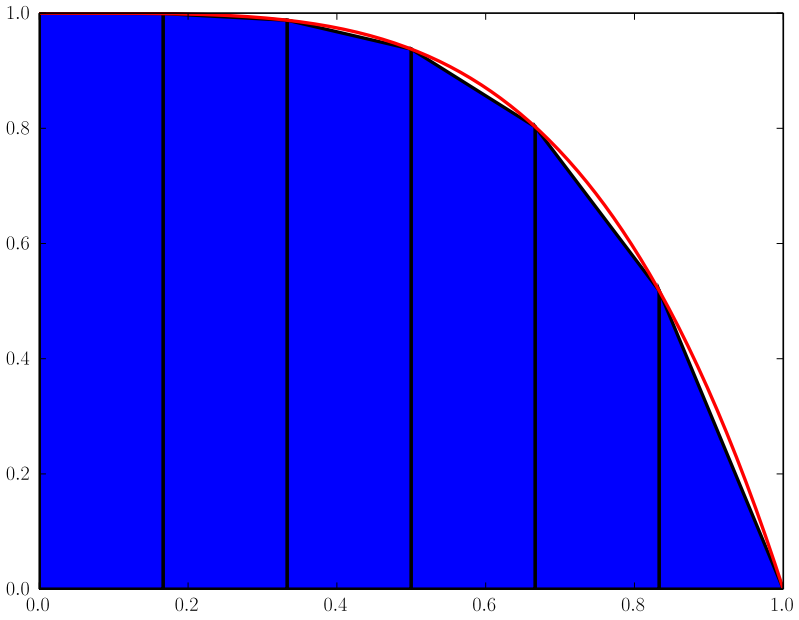

Figure 17.2: Demonstration of approximating the integral $\int_0^1 (1-x^4)dx$ using a composite trapezoidal rule. The shaded area is the area approximated by the composite rule, and the actual function is given in red.

that aren't either a or b get counted twice because they belong to sub-intervals to their left and right.

Problem 2. Implement Simpson's, Simpson's 3/8 and Boole's composite rules as Python functions. Require the user to input the number of sub-intervals. Now test your function by integrating $x^{1/3}$ on the interval $[0, 1]$ (which you can analytically solve).

One other important technique is what is known as *Adaptive Quadrature*. It is clear in Figure 17.2 that we have approximated the integral better in some places than in others. In fact on the left side of the interval the approximation by linear interpolants is almost indistinguishable from the actual function, while on the right the approximation clearly introduces more significant errors. Adaptive Quadrature methods attempt to identify locations where the error is bad and subdivide only those areas.

One way to approximate the error of a particular integral is to use a composite rule. For example (is we let $S([a, b])$ denote the output of Simpson's rule on a specific interval)

$$\text{Error}(S([a,b])) \approx |S([a,c]) + S([c,b]) - S([a,b])|$$

Where $c = \frac{a+b}{2}$. We can then decide if the error is appropriately small, and split the interval if it is not. By repeating this process recursively we can achieve the higher accuracy of a very fine composite quadrature rule while only evaluating the function in areas of interest.

This error indicator is very crude, and many other indicators have been proposed. However, they can become complicated relatively quickly, so we won't discuss them here.

Problem 3. Write a function that uses an adaptive version of Simpson's rule. The user should input the maximum error. It should use a recursion. You will have to divide the error between sub-intervals.